System Design Document for Source Code Analysis Tool

Oğuzhan Yangöz, Ashwin Kishore, Yitao Wei

CSCI 6231 Software Engineering

May 4, 2022

1. INTRODUCTION

This System Design Document (SDD) was developed as part of the Source Code Analysis Tool project, in parallel with System Requirements Document (SRD) dated March 11, 2022.

1.1 Purpose

The primary purpose of this System Design Document (SDD) is to describe the architecture and system design of Source Code Analysis (SCA) tool through various diagrams, including class hierarchical diagrams and a sample interaction diagram, along with the descriptions. The intended audience of this document is the current and future members of the software development team and project managers.

1.2. Scope

The Source Code Analysis (SCA) tool project was designed to allow project managers and developers to document and keep track of their repositories as they evolve into larger projects over time. Furthermore, Source Code Analysis (SCA) tool aims to assist the newly recruited developers and engineers with their acquaintance with the relevant repositories. Being the second part of the three-stage process, this SDD document focuses on how the requirements listed in System Requirements Document will be satisfied by detailing the system architecture.

1.2.1. Design Constraints

This System Design Document outlines the preliminary implementation plan of the Source Code Analysis (SCA) tool and is subject to change at any stage of the project development if deemed necessary.

1.3 Overview

This System Design Document (SDD) presents how the Source Code Analysis (SCA) tool will be implemented starting from the overall system architecture and moving to discrete components of the system. For visualization purposes, UML diagrams were included along with the textual descriptions.

1.4 Definitions and Acronyms

SCA: Source Code Analysis –the project being developed that is discussed in this document.

SRD: System Requirements Document – describes the user requirements, system requirements, functional and non-functional requirements of the SCA project (Kaisler, 2022).

STT: System Test Document –provides descriptions and steps of testing procedures for the software system to check whether the project meets the requirements in SRD.

Process: The process of interaction starting with the smallest element of a system to the larger components.

Object: An instance of an Object-oriented design, which represents real-world entities.

Class: A piece of extendable code template that is used for declaring variables, their initial values and methods for the intended behavior.

Attribute: A specification of class with a specific data type and structure.

Data type: A set of built-in types for variables such as Boolean, Float, Double, Boolean, Integer and Array.

Method: A set of operations that is executed upon a call through global or local parameters.

Component: A collection of classes that provides a certain functionality of a system.

Module: A collection of components within a system or subsystem to provide certain functionality.

Subsystem: A large collection of modules to implement a large functionality. (e.g. Data Entry subsystem)

System: An organized collection of modules/subsystems that create a fully functional piece of software by interacting its subsystems efficiently in line with the customer requirements.

Repository: A container or storage that stores a chunk of data.

Session: The time elapsed between the last log-in and log-out of a user.

2. SYSTEM OVERVIEW

The Source Code Analysis (SCA) tool will have the following subsystems;

- User Interface subsystem
- Query subsystem
- Report subsystem
- Data entry subsystem
- Functional analysis subsystem.

2.1 User Interface Subsystem

User Interface subsystem will be responsible for providing a UI, functionalities and tools required for users' interaction with the software. User Interface subsystem will also include file management, displaying information on a specific part of a repository, an interactive command line interface as well as window management within the SCA tool. This subsystem, like all subsystems, will closely interact with the other subsystems to provide users with full functionality. (

2.2 Query Subsystem

Query subsystem will be responsible for the interaction through queries and retrieving tables and information. By leveraging the tight integration of command line interface and other User Interface components, Query subsystem will allow users to perform various types of analyses, aiming to assist users with the hierarchical and architectural structure of the project, as well as the content of the repository.

2.3 Report Subsystem

Report subsystem will be responsible for generating a report including description of systems, subsystems, modules, classes, methods and attributes. In order to list those elements by order, a tree and a flame diagram will be created to organize those data and descriptions. Report subsystem will allow users to filter data to get what they are interested in. Besides, the report subsystem will allow users to export reports with selectable formats such as pdf, doc, and xls.

2.4 Data Entry Subsystem

Data entry subsystem will be responsible for adding, deleting and modifying most elements in the system such as system, subsystem, module, class, method, attribute, interface. Before implementing add, delete and modify operations, repetition and format check will be performed first to ensure the entered command meets the format requirements. Besides, the subsystem will capture and record each variable, its data type, and description during system running.

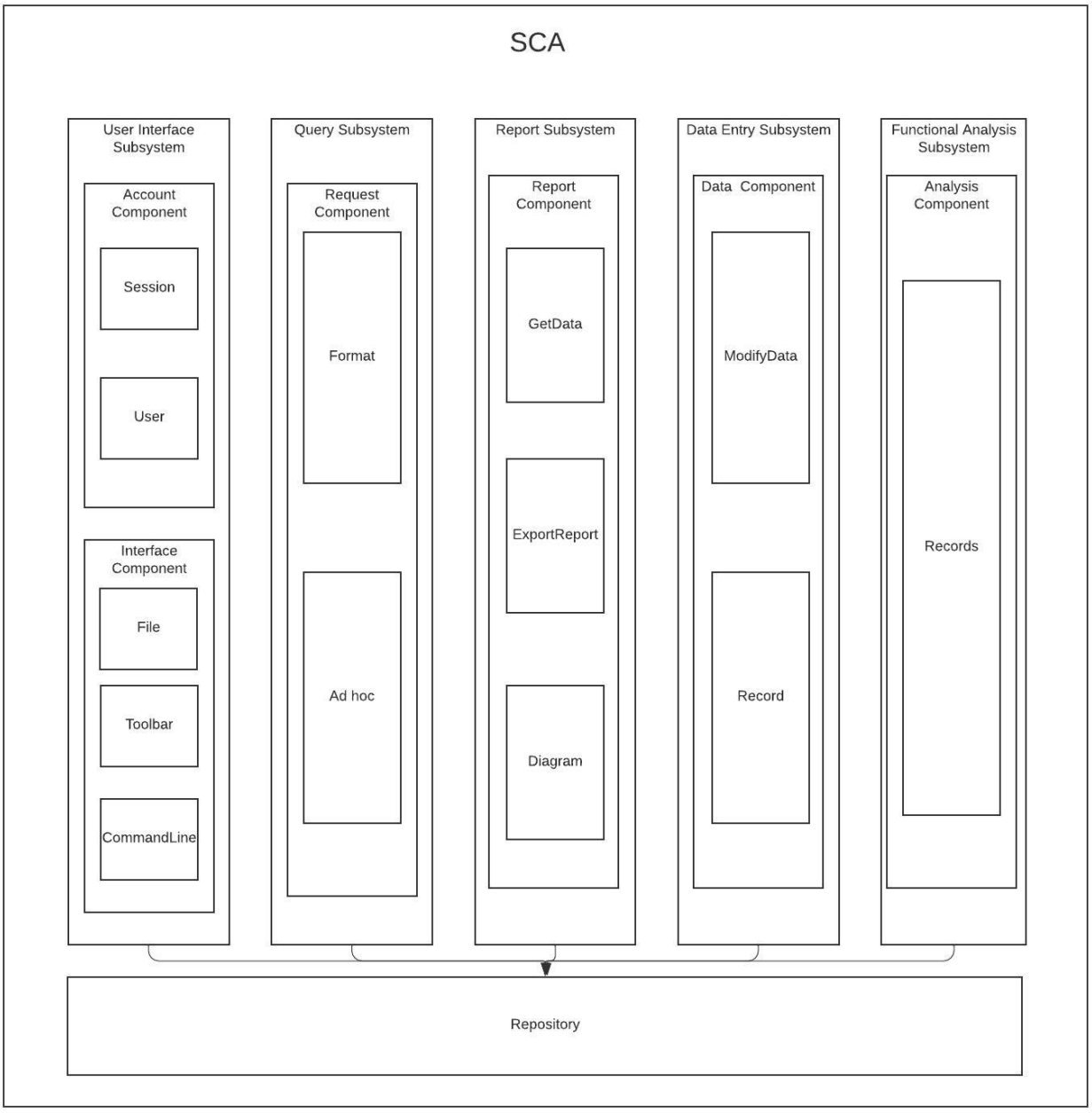
2.5 Functional Analysis Subsystem

The Functional Analysis Subsystem will be responsible for keeping track of decomposition in a relational UML structure. It lets users access all the records that are required in the Data Entry System,attributes, their values, data type and also allow users access to the return type of all method, all the classes defined in the module,access the global values from all classes within the module

3. SYSTEM ARCHITECTURE

The high-level system architecture of the Source Code Analysis (SCA) tool is as shown in Figure 1 below;

Figure 1: Architecture Diagram (adapted from the architecture diagram in the SRD document)



3.1. System Architecture Description

Source Code Analysis (SCA) tool consists of 5 subsystems: User Interface, Query, Report, Data Entry and Functional Analysis.

3.1.1. User Interface Subsystem

User Interface subsystem includes the following components: Account and Interface. (SRD 6.1)

3.1.1.1. Account Component

Account component was designed to provide infrastructure for user log-in, registration and keeping session information and logs of the users through User and Session classes.

3.1.1.2. Interface Component

Interface component is responsible for providing a user-friendly graphical user interface to all users of the SCA tool by providing toolbar functionality which aims to assist users with editing files within the repository.

In addition, Interface component provides users with a command-line interface where users can run the commands to interact with repositories.

Any functionality provided by Interface component relies on the File class to interact with the files and perform operations such as opening and closing a file.

3.1.2. Query Subsystem

Query subsystem includes a Request component and is responsible for allowing interaction between the database and the SCA tool. (SRD 6.2)

3.1.2.1 Request Component

Request component consists of Format and Ad-hoc classes, which provides users with functionalities such as making specific types of queries, and ad-hoc queries. The component validates the queries and allows users to save or update the database.

3.1.3. Report Subsystem

Report subsystem includes Report component and is responsible for ensuring users can get different types of data from the target repository, export a report in csv or any other predefined format and visualize the requested data. (SRD 6.3)

3.1.4. Data Entry Subsystem

Data Entry subsystem has a Data Component which is responsible for allowing users to enter data through commands or user interface. Some sample functionalities of Data Entry subsystem are as follows;

- Allowing users to update the objects in the repository
- Allowing users to assign methods and attributes to an object as well as providing a pseudocode description of a method that specifies its input parameters and return type. (SRD 6.4)

3.1.5. Functional Analysis Subsystem

Functional analysis subsystem includes Analysis component, and is responsible for mapping out the class structure, performing attribute, method and module analysis of a repository. Some of the sample functionalities are as follows;

- Recording the result type for each method or void
- Creating an entry for each of a class
- Capturing the required input parameters for a method (SRD 6.5)

3.2 Design Rationale

While designing the architecture of the Source Code Analysis (SCA) project, a bottom-up approach was used. The rationale behind this decision is as follows;

- Potentially higher success rate in satisfying the user requirements in System Requirements Document and easier adaptability to making changes as the project evolves.
- more effective and optimal decision-making process for a team of developers and engineers despite taking longer to decide.
- Easier for many developers to work on different components of the project simultaneously.
- Easier to apply decomposition over inheritance principle

4. DATABASE AND DATA DESIGN

In the Source Code Analysis (SCA) project, logical data models will be used as they are useful for giving precise information about the process and relationships between the entities as well as providing information regarding the required input for the processes.

4.1. Database Layer

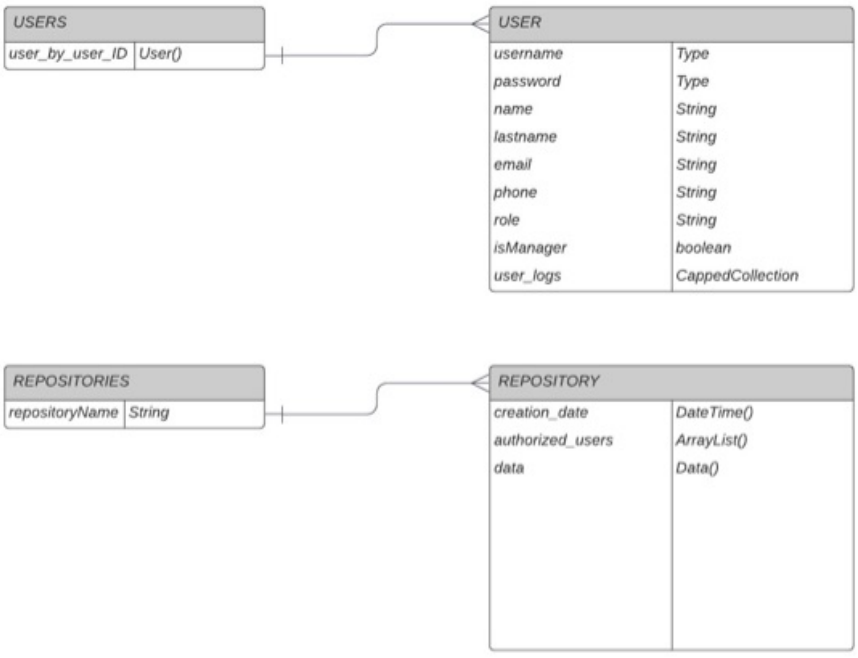
MongoDB will be used as the database to store and retrieve data primarily because it is a document-database and it offers high flexibility in terms of data structure variations. Furthermore, MongoDB is highly scalable and appropriate for the use of developer teams.

4.2. Data Model

MongoDB supports both structured and unstructured data, and this gives a lot of flexibility to the design team in many ways.

This project will use json format to store all data components of the SCA tool.

Figure 2:Preliminary Sample Data Model



The modules, components, subsystems, classes, attributes, variables, functions/methods of a repository will be stored in the data attribute.

4.2.1. Description of the Data Model

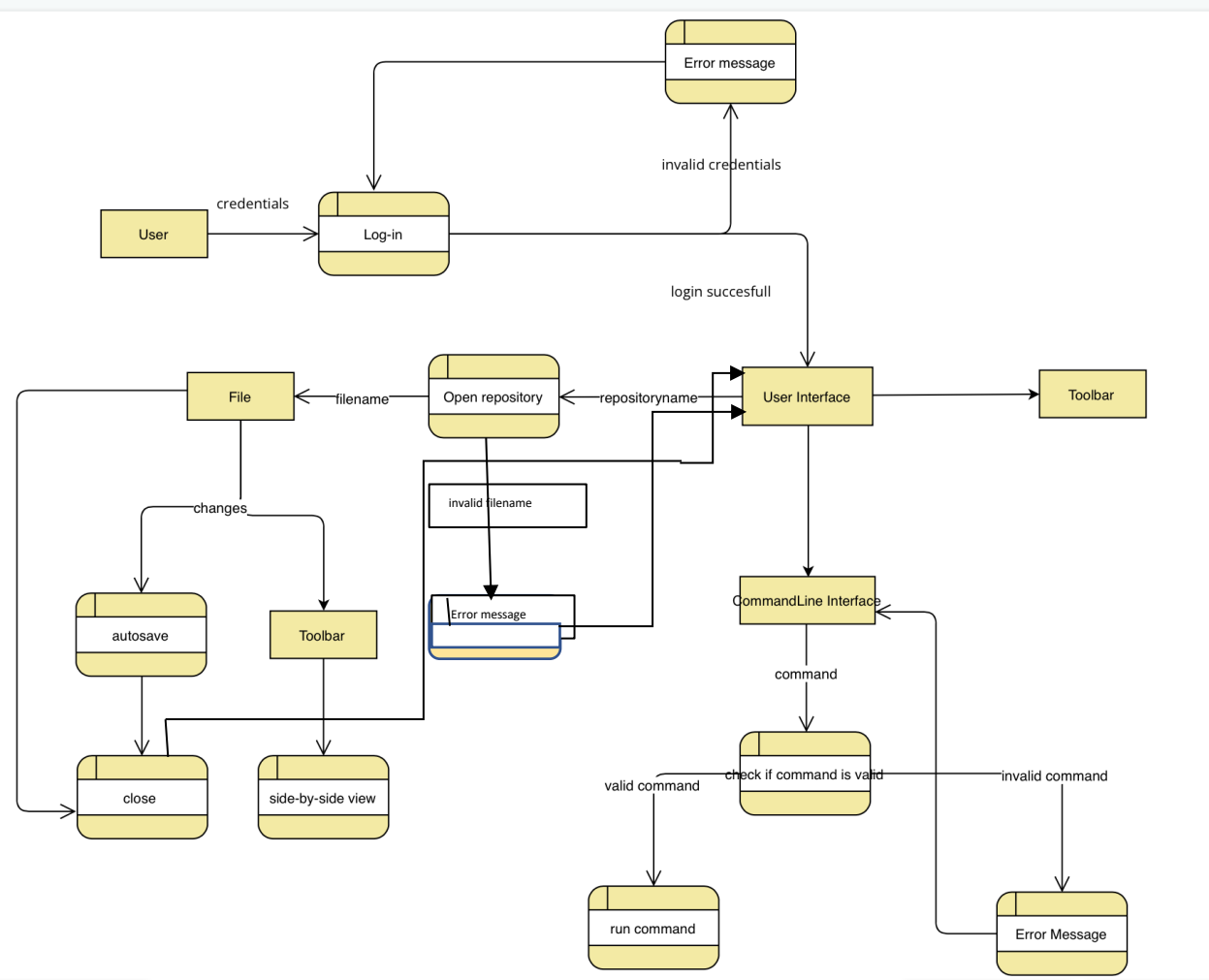
As shown in Figure 2, user data is stored by user_id, and user information is stored inside the user_id, where all user details including credentials and user logs are stored.

Similar to users, repositories are also classified by repositoryName, and all details about the repository are stored inside it. Data sub branch in each repository stores all the main project details such as classes, subsystems, variables, functions and their descriptions. The diagram does not outline the architecture of the data sub branch.

4.3. Sample Data Flow Diagram

A sample data flow diagram is as shown in Figure 3 below along with its description.

Figure 3: Sample Data Flow Diagram



4.3.1 Sample Data Flow Diagram Overview

Figure 2 illustrates sample data flow for the following functionalities;

- Logging in to the SCA tool through the login panel,
- Performing file operations such as closing, autosaving or viewing multiple files side-by-side,
- Running a command through command line interface,

4.3.2. Sample Data Flow Diagram Elements

In Figure 8, four different UML elements were used;

- Solid Yellow Rectangular UML elements: They represent classes or components within a subsystem.

- White/Yellow Rectangular UML elements: They represent methods to be called for the intended operation.

- Arrows: Arrows demonstrate the flow and order of operations to be performed along with the source and target.

- Mid-arrow text elements: They represent input to the target method to be called.

4.3.2. Sample Data Flow Diagram Description

In this section, the description of each functionality illustrated in Figure 8 is provided.

4.3.2.1. Logging in

⇒ A register user of the SCA tool enters their credentials in the log-in screen (username and password)

⇒ When the user pressed the log-in button, the log-in method is called.

- If no credentials are entered for the username and password input areas, the method displays a pop-up window stating "*No credentials entered. Username and password areas cannot be left blank* ".

⇒ The entered credentials are sent to the database and checked if there is a match for the username and password combination.

- If the entered credentials fully match a user's credentials in the database, User Interface Main Screen (Figure 11) is displayed, and the user's session is initiated successfully.

- If the entered credentials are not recognized by the database, the method displays a pop-up window stating "*Invalid username or password. Please check your credentials and try again.*"

4.3.2.2. Performing File Actions

⇒ Following the successful log-in process, the user will have access to file menu, command-line interface, and toolbar (See Figure 11 for the mock model).

⇒ Using the file menu, the user will be able to view the accessible repositories in the database. By choosing the relevant repository from the drop-down menu or typing the repository name into the tiny search bar in the first row of the drop-down menu, openRepository method will be called.

- This method will trigger an internal method;
  - checkIfRepositoryExists: This will check if there is a repository in the database that matches the string name entered as an input by the user.
    - If there is a match, the method will return true and the execution of openRepository will continue.
    - If there is no match, the repository will display a pop-up window stating "*Repository not found. Please check the repository name and try again*. ". Upon clicking the "OK" button in the pop-up window, the user will return to the main screen.

⇒ The user will have access to files within the repository and make a selection from the drop-down menu that displays the files within the repository.

⇒ The file will be accessible by the user. At this stage, users will be able to edit the file, close it or change the view option using the toolbar.

- If the user makes changes, the autosave method will be called automatically and it will overwrite the most recent version of the file. The method will also store the latest individual changes to allow for the command + Z functionality, which will be triggered by the undo method if necessary.
- If the user closes the file using the window manager close icon, the file will be closed, and the user will return to the main user interface screen.

- If the user clicks the toolbar on top when the selected file is being displayed and clicks side-by-side view icon, the user will be prompted to choose another file within the same repository from the drop-down menu and the selected file will be displayed in the half of the file-view screen in the Figure 11 along with the 1st file. A maximum of two files will be supported in side-by-side-view functionality.

4.3.2.3. Running a Command

⇒ After a successful log-in, if the user enters a command through the command line interface on the main screen, the user will be able to enter a command.

- When a command is entered by the user, checkIfCommandIsValid internal method will check whether the command is valid and recognized by the system. If it is invalid or the command cannot be executed at that time, it will print an error statement. Based on the type of the error, the command line will print a customized error message and allow the user to stay in the command line interface ready for a new input. Error types and error messages for the command line interface are as follows;
  - 001 Invalid command – Command not found.
  - 002 Invalid directory – Directory not found
  - 003 Invalid filename – File not found.
  - 004 Connection Error – Connection to the database lost
  - 005 Authorization Error – User not authorized (e.g., manage vs developer)
- If the entered command is valid and no error message is displayed, the command will be run and perform the target task of the command.

5. COMPONENT DESIGN

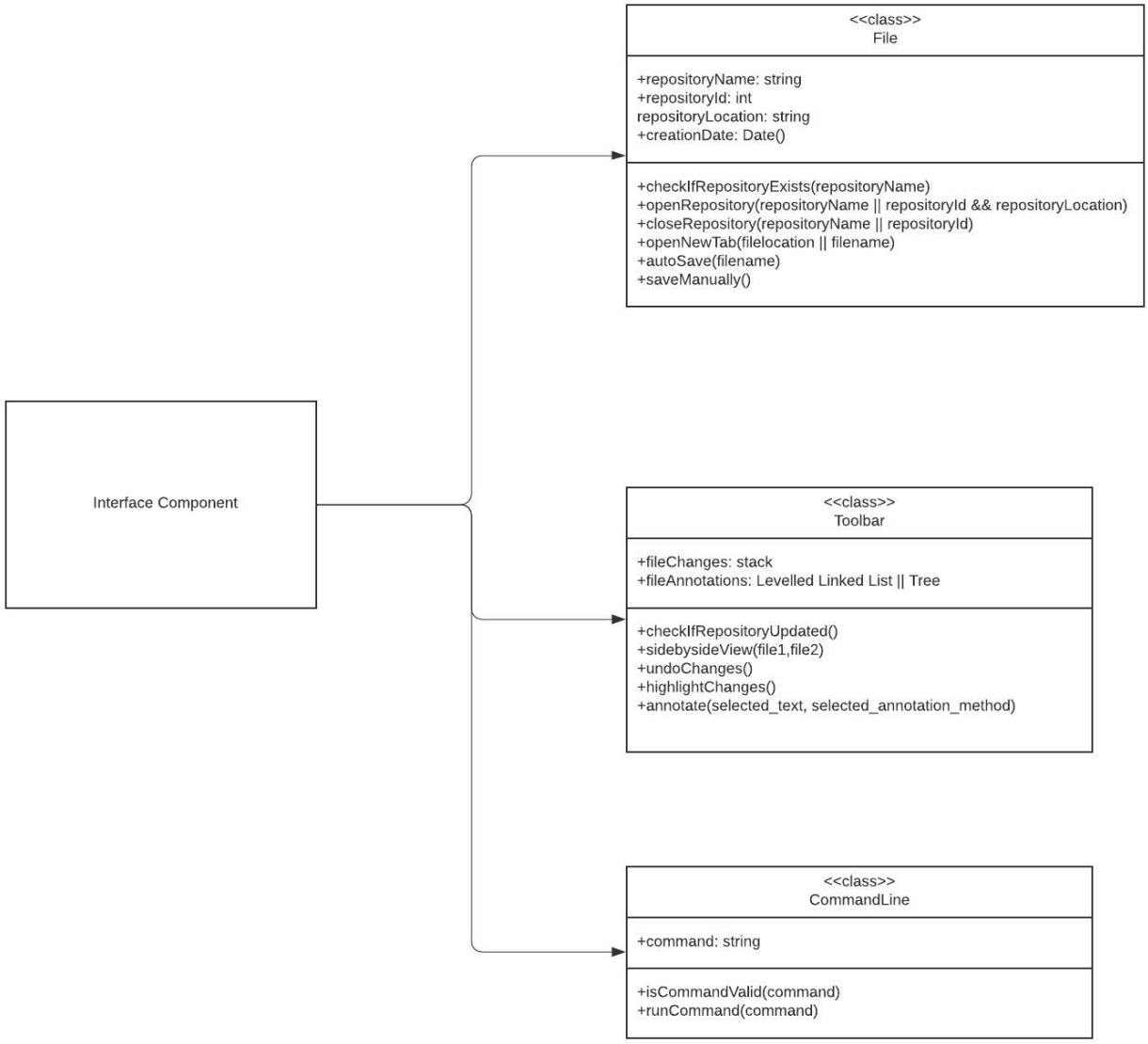
This section provides short descriptions of the methods and the diagrams of each component.

*Getters, setters and constructors will be created via the Lombok module in each component.

5.1. Interface Component Descriptions

The methods associated with the interface component will be as follows.

Figure 4: Interface Component Diagram



5.1.1. File

- checkIfRepositoryExist: It checks whether the input repository name exists and returns true or false. If the repository does not exist, the method prints "*Repository not found*" and returns false, which stops the execution of the outer method. If the repository exists, the method does not raise any error and returns true.
- openRepository: It opens the input repository after internally running checkIfRepositoryExists method. If it returns true, the function will display the file opened

in the file view section of the user interface. If it returns false, the error message described in checkIfRepositoryExist method will be displayed

- closeRepository: it closes the repository after internally running the autosave method and checking if there is any interrupting operation running. If the execution is stopped because of the autosave method or closeRepository method itself, it displays a pop-up window stating "*SCA tool is unable to close the repository. Check your internet connection to autosave and close the file as expected.*"
- openNewTab: it opens another file in a new tab by running two tasks simultaneously after checking if the new target tab is valid and the tool supports the file type. If it is not valid, it displays a pop-up window stating "*File cannot be opened. Please try a different file format.*". Supported file formats are; pdf, txt, csv, source code files(cpp,.java,.py …)
- autosave: The method checks if the file is still being edited and any user is still making changes. Whenever a change is made in the document, the changes are saved, and the file is overwritten automatically. If there is any connection error that prevents the autosave method from connecting to the database, the method displays a pop-up window stating "Connection error. Please check your internet connection."

5.1.2. Toolbar

- checkIfRepositoryUpdated: It checks whether there is a change in the repository after running checkIfRepositoryExists method. If the internal method returns true, the method is executed. If the repository is updated it returns true, otherwise, it returns false.
- sidebysideView: It checks if the second target file is supported by the tool. If supported, it opens two files simultaneously and divides the screen size of the file view section and toolbar into two. If the second file cannot be opened, it displays a pop-up window stating "*File not supported. Please try a different file format."* A maximum of two files are supported by the method. If the user attempts to view three files in the file view, the method displays a pop-up window stating, "*You can only view two files side-by-side".*
- undoChanges: It checks if there were any changes made after the file was last opened. If there are, it pops the last element of the stack and removes the most recent change upon pressing the undo icon in the toolbar or Command + Z combination.

- highlightChanges: The method is activated upon pressing the highlight button in the toolbar. It sets the background color of the changes made after the last saved version of the repository to yellow by comparing the most recently saved version and the most recently displayed version. If there are no changes, the method displays a pop-up window stating, "*No changes detected."*
- annotate: The method is activated through the icon on the toolbar by the user. The method creates a cloud-shaped comment input box with an arrow pointing to the selected text. The text field in the input box is updated and displayed directly upon the return key. If the text field is left blank, the input box is automatically removed. The input box size is adjusted relative to the text size, which has no upper bound.

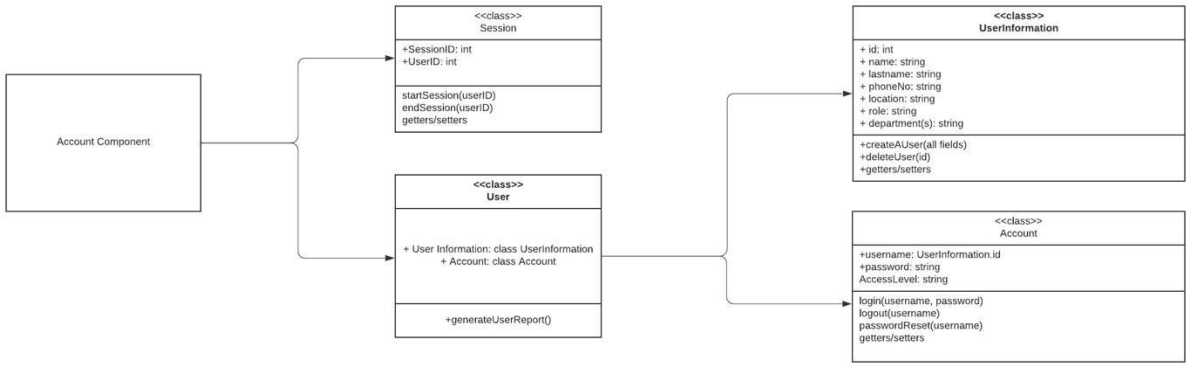
5.1.3. CommandLine

- isCommandValid: It checks if the entered command is valid. If it is not, it returns one of the following errors depending on the error type: *"001 Invalid command – Command not found", "002 Invalid directory" – "Directory not found" ,"003 Invalid filename – File not found", "004 Connection Error – Connection to the database lost", "005 Authorization Error – User not authorized (e.g., manage vs developer)*".
- runCommand: It runs isCommandValid method internally and if the return value is true, it executes the command. If the internal method returns false, it stops execution and prints the internal method's error message.

5.2. Account Component Descriptions

The methods associated with the account component are as follows;

Figure 5: Account Component Diagram



5.2.1. Session

- startSession: It is triggered when the login method returns true, and it starts the timer and keeps track of the time elapsed until the endSession method is successfully executed or the SCA tool is shut down. The registered user's username and id are also recorded. Current information is displayed and updated on the session information section in the main screen.
- endSession: It is triggered when the logout method is invoked or the SCA tool is shut down. It checks whether there is a session running. If there is no ongoing session, the method raises "No session found". It stops the timer and saves the session id, start time, end time, time elapsed (h/m/s) and user id and username into the database. If there is any connection error, it displays a pop-up window stating "*No internet connection. Session could not be terminated*".

5.2.2. User

- generateUserReport: it creates a table with the columns of user id, name, last name, role, department, session_id, session duration, session start, session end, and exports it as an excel file. It populates the created columns and rows with the session information of the user from the database, that can be accessed via user_id. If the user_id is not registered in the system, it displays a pop-up window stating, "*User not found*". If the user_id is registered but there is no session information recorded, it displays a pop-up window stating, "*User does not have enough data to generate a user report*".

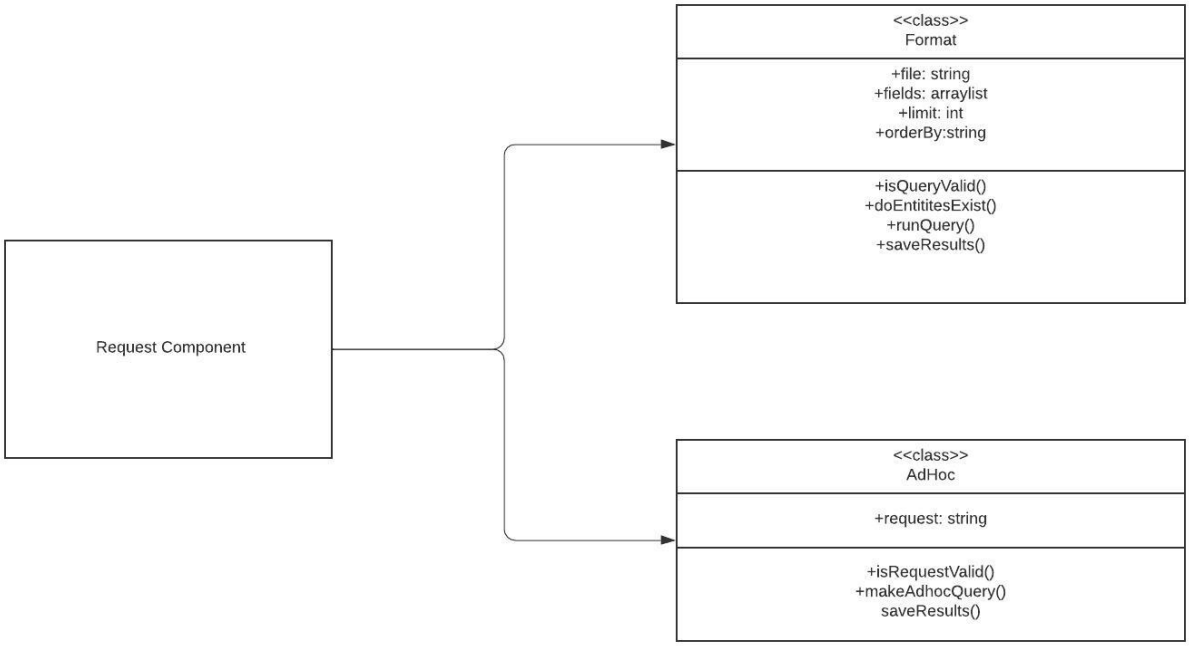
- createAUser: It creates a new user object using all the fields, grants authorization to the select repositories and saves it to the database. It can only be called by users that are assigned the role “Manager”. Users without a Manager role cannot view the option in their user panel.
- deleteAUser: It deletes the user from the database using the user id. If the user_id is not found, it displays a pop-up window stating, “User_id/User not found. Task could not be completed”. If the method is triggered through the command line interface, it prints the above error message instead of a pop-up window. Restricted to Manager-role users only.
- login: It checks whether the username and password entered correctly and fully match the credentials in the database.
    - If no credentials are entered for the username and password input areas, the method displays a pop-up window stating “*No credentials entered. Username and password areas cannot be left blank* “.
    - If the entered credentials fully match a user’s credentials in the database, User Interface Main Screen (Figure 11) is displayed, and the user's session is initiated successfully.
    - If the entered credentials are not recognized by the database, the method displays a pop-up window stating “*Invalid username or password. Please check your credentials and try again.”*
- logout: When triggered, it ends the session by running endSession method internally and redirects users to the login page, removing access to the main page. When triggered, the method first checks if the user calling the method is logged in. If the user is not logged in, it displays a pop-up window stating “*Failure. User is already logged out.”.*
- passwordReset: The method takes username or/email address as an input and sends an SMTP password reset email to the user’s registered email address with a link to update the password after checking that the entered username/email exists in the user database. Regardless of the outcome of the verification, a pop-up message saying “*If the username/email matches our records, we will send you a password-reset email shortly.*” upon pressing the “Reset” button. The newly entered password entered through the link

will replace the previous password in the database upon successful reset. The link in the email will expire in 30 minutes.

5.3. Request Component Descriptions

The methods associated with the request component are as follows.

Figure 6: Request Component Diagram



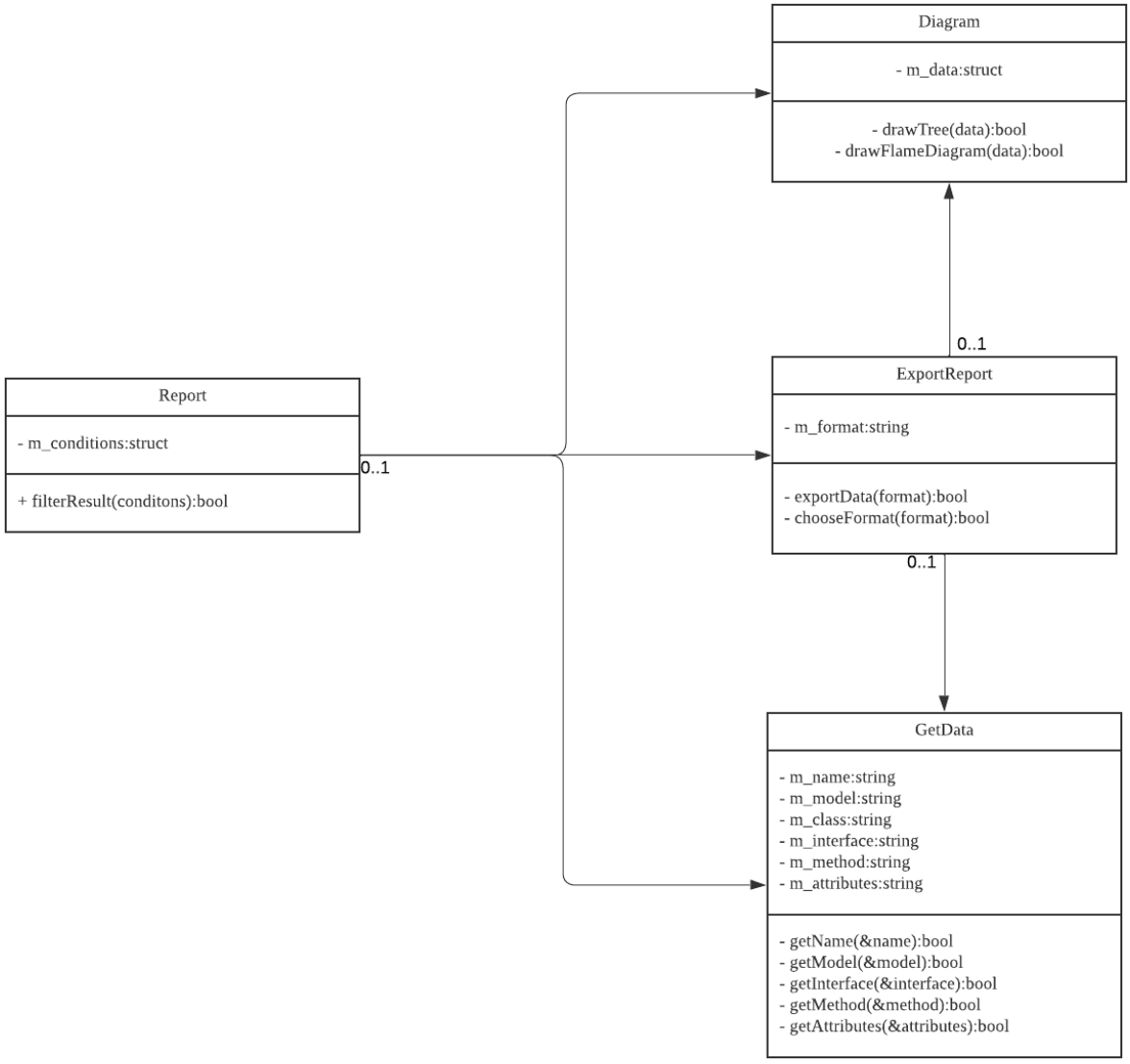
- isQueryValid: The isQueryValid function checks whether the given query is syntactically correct. If it is syntactically not correct, it returns false, and also checks if the commands given match. if the command(s) do not match, it returns false. The returned value is always boolean. There is also a pop-up dialog to report the error.
- doEntitiesExist: The doEntitiesExist function checks the entire database for the data that is entered by the user. After checking the entire database if the entities entered are not found in the repository the function returns false and pops up a dialog to report the error message.
- runQuery: The runQuery function runs the isQueryValid method internally and validates the query given by the user and returns a boolean value, if it returns true, it executes the commands. Else it returns false and pops up a dialog to report the error message.

- makeAdhocQuery: The makeAdhocQuery function takes the command from the user and checks the syntax if the user has entered it correctly or not if it does not match the function returns false and pops up a dialog to report the error message.
  if the syntax is valid it executes the commands.
- saveResults: The saveResult function after all the validation when saveResults function is called it adds the result table to the database. If after the execution, it pops up the error message if any error occurs such as "*Database not found*", else it displays the success message.

5.4. Report Component Descriptions

The methods associated with the interface component will be as follows.

Figure 7: Report Component



- filterResults: It filters the record data by the conditions users select. The conditions are as follows: time, subsystem, module, class, method, and interface. Once users set their interested conditions, this function will query the database and get the data with those interested parts. Then this subsystem will call drawTree() and drawFlameDiagram() functions to update the diagram(SRD 4.3.4).
- getName: This is a private function called by other functions when they need to get all names for subsystems, modules, and classes. This function will query the database and

return all names without filtering. If the name doesn't match the name found in the database, this function will return false and pop up a dialog to report the error(SRD 6.3.1).

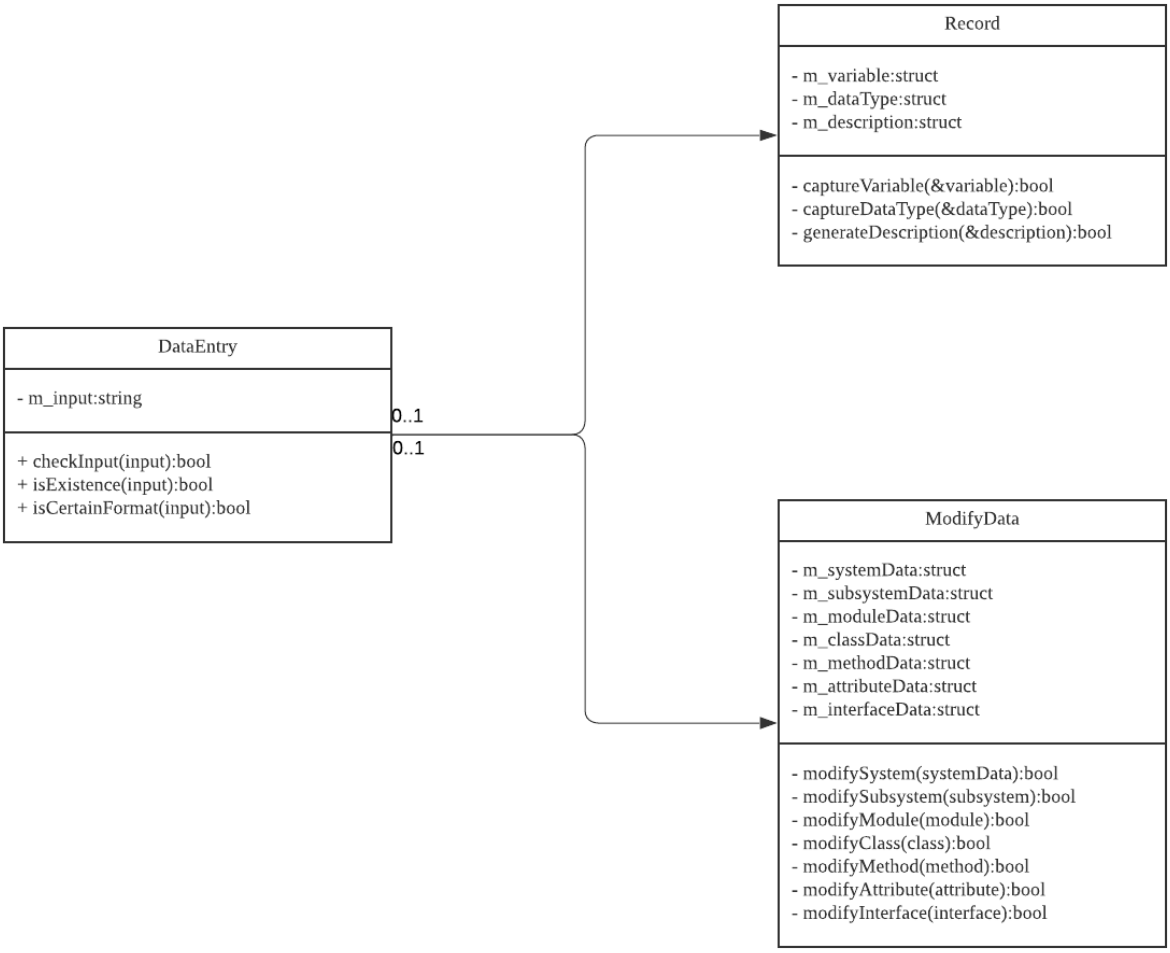
- getModel: This is a private function called by other functions when they need to get the structure of a module in terms of classes. The parameters of this function are a vector of module names that can be obtained from the getName() function. This function will query the database and return the structure in terms of classes corresponding to the name of modules. If the name doesn't match the name found in the database, this function will return false and pop up a dialog to report the error(SRD 6.3.4).
- getInterface: This is a private function called by another function when they need to get interfaces when the user specifies a class. The parameter of this function is a vector containing the names of classes which can be obtained from the getName() function. This function will query the database and return the interfaces corresponding to the name of classes. If the name doesn't match the name found in the database, this function will return false and pop up a dialog to report the error(SRD 6.3.2).
- getMethod: This is a private function called by another function when they need to get methods when a user specifies a class. The parameter of this function is a vector containing the names of classes which can be obtained from the getName() function. This function will query the database and return the methods corresponding to the name of classes. If the name doesn't match the name found in the database, this function will return false and pop up a dialog to report the error(SRD 6.3.3).
- getAttributes: This is a private function called by another function when they need to get attributes when a user specifies a class. The parameter of this function is a vector containing the names of classes which can be obtained from the getName() function. This function will query the database and return the attributes corresponding to the name of classes. If the name doesn't match the name found in the database, this function will return false and pop up a dialog to report the error(SRD 6.3.3).
- exportData: This is a private function called by the user when they decide to export the data. First, the function reads the current data from the diagram and processes the data to a certain format. Then the export report is generated(SRD 4.3.3).

- chooseFomat: This is a private function called by exportData() function when the data needs to be processed to a certain format such as pdf, doc, etc. The parameter of this function is the name of a format like pdf. Each format has its own template. So this function will call the template with the name of the format and fill the template with the data. If the name doesn't match the name of the template, this function will return false and pop up a dialog to report the error(SRD 6.3.7).
- drawTree: This is a private function called when users decide to review the results. This function will call the functions introduced above to obtain the data with or without filtering and draw a tree of systems, subsystems, modules, classes, and methods by recording data(SRD 6.3.9).
- drawFlameDiagram: This is a private function called when users decide to review the results. This function will call the functions introduced above to obtain the data with or without filtering and draw a flame diagram of systems, subsystems, modules, classes, and methods by recording data(SRD 6.3.9).

5.5. Data Entry Component Descriptions

The methods associated with the interface component are as follows.

Figure 8: Data Component



- checkInput: This is a private function used to check if entered data is repete or meets format requirements. Once the user finishes typing the command, this function will first inspect if the command meets the format requirements by calling isCertainFormat() function. Then the function will check repentance by calling the isRepete() function in order to avoid repeat objects(SRD 6.4.1).
- isExistence: This is a private function used to check if entered data exists. The function will compare the input with the data exited in the database. For the add operation, object existence will be checked first. If not, add operation is performed, otherwise is unsuccessful and pops up the warning dialog. For the delete and modify operations, if the

object exists, the operation is successful otherwise is unsuccessful and a warning dialog(SRD 6.4.1).

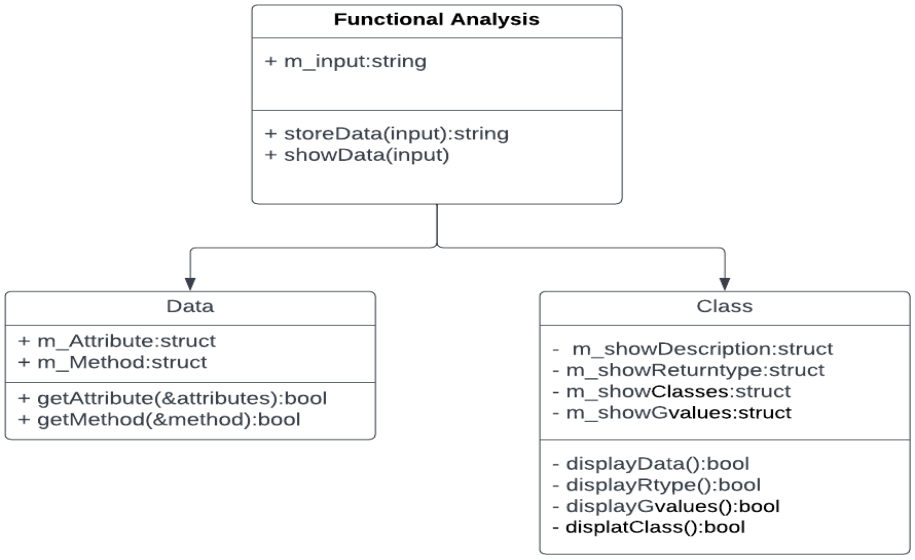
- isCertainFormat: This is a private function used to check if entered data meets certain criteria. This function will inspect if the command meets the format requirements by regular expression. If it does exist, returning is true, otherwise is false(SRD 6.4.2).
- modifySystem: This is a private function used to implement add, delete or modify operations to systems. Before the implementation, the isExistence() function will be called first to guarantee whether the object exists or not. Then if the implementation is unsuccessful, a warning dialog will be shown to let the user know the error(SRD 6.4.3).
- modifySubsystem: This is a private function used to implement add, delete or modify operations to subsystems. Before the implementation, the isExistence() function will be called first to guarantee whether the object exists or not. Then if the implementation is unsuccessful, a warning dialog will be shown to let the user know the error(SRD 6.4.3).
- modifyModule: IThis is a private function used to implement add, delete or modify operations to modules. Before the implementation, the isExistence() function will be called first to guarantee whether the object exists or not. Then if the implementation is unsuccessful, a warning dialog will be shown to let the user know the error(SRD 6.4.3).
- modifyClass: This is a private function used to implement add, delete or modify operations to classes. Before the implementation, the isExistence() function will be called first to guarantee whether the object exists or not. Then if the implementation is unsuccessful, a warning dialog will be shown to let the user know the error(SRD 6.4.3).
- modifyMethod: This is a private function used to implement add, delete or modify operations to methods. Before the implementation, the isExistence() function will be called first to guarantee whether the object exists or not. Then if the implementation is unsuccessful, a warning dialog will be shown to let the user know the error(SRD 6.4.3).
- modifyAttribute: This is a private function used to implement add, delete or modify operations to attributes. Before the implementation, the isExistence() function will be called first to guarantee whether the object exists or not. Then if the implementation is unsuccessful, a warning dialog will be shown to let the user know the error(SRD 6.4.3).

- modifyInterface: This is a private function used to implement add, delete or modify operations to interfaces. Before the implementation, the isExistence() function will be called first to guarantee whether the object exists or not. Then if the implementation is unsuccessful, a warning dialog will be shown to let the user know the error(SRD 6.4.3).
- captureVariable: This is a private function used to capture information about variables and generate descriptions during system running. If there is an error capturing a variable, the function will skip this variable and make a record in the log which will be shown after running(SRD 6.4.6).
- captureDataType: This is a private function used to capture information about data types and generate descriptions during system running. If there is an error capturing a variable, the function will skip this variable and make a record in the log which will be shown after running(SRD 6.4.6).
- generateDescription: This is a private function used to generate descriptions for captured elements including data and time stamp associated with its creation in the repository and the last time the object was updated(SRD 6.4.7).

5.6. Functional Analysis Component Descriptions

The methods associated with the interface component are as follows.

Figure 9: Functional Analysis Component



- storeData: The storeData method stores the data from the user and creates an entry for each attribute of a class and records the data type of each attribute in a class. If there is no entry the error message pops up saying there is no data available.
- showData(input):The showData method allows the users to access all the records that are required in the Data Entry System. If there is no record available it pops up an error message stating that there is no data to be displayed.
- getAttribute(): The getAttribute method returns all the attributes by data type with respect to the user's input and it also returns the descriptions for each attribute. Then if the implementation is unsuccessful, a warning dialog will be showed to let the user know the error.
- getMethod(): The getMethod function analyzes the methods with the list of variables that the method uses. It throws an error if a void result is returned.
- displayData(): It allows users access to all the classes defined in the module. If the method is empty it will return false and it pops up the error message if this error occur (like not found…) else it displays "Class is defined in the system"
- displayGvalues():It allows users access the global values from all classes within the modul . If the method is empty it will return false and it pops up the error message if this error occur (like not found…) else it displays the success message
- displayRtype(): It allows users access to the return type of all methods. If the method is empty it will return false and it pops up the error message if this error occurs (like not found…) else it displays "Method is defined in the system"

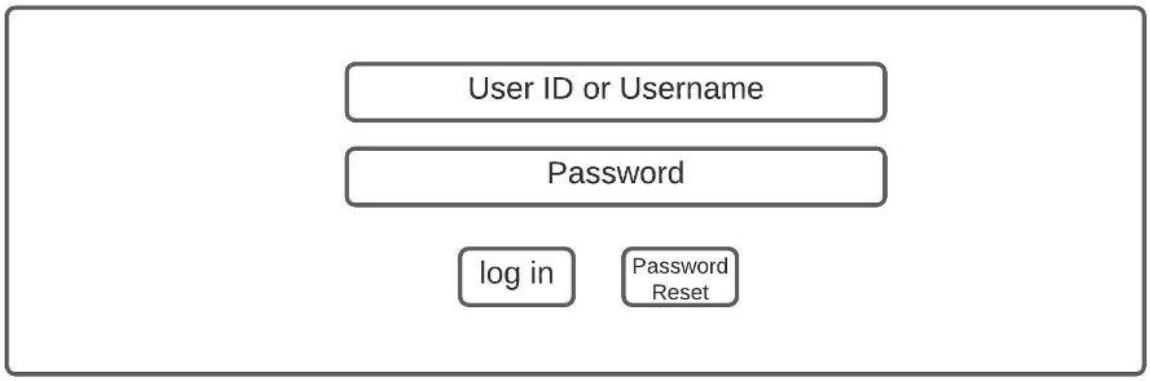
6. HUMAN INTERFACE DESIGN

The human interface design of the Source Code Analysis (SCA) tool will be powered by the User Interface subsystem. User Interface will be designed using Java FX.

6.1 Log-in Panel

Preliminary design of the user interface is as shown in Figure 9: User Log-in Panel and Figure 11: User Interface - Main Screen Layout.

Figure 10: User Interface - Main Screen Layout (adapted from the SRD document)

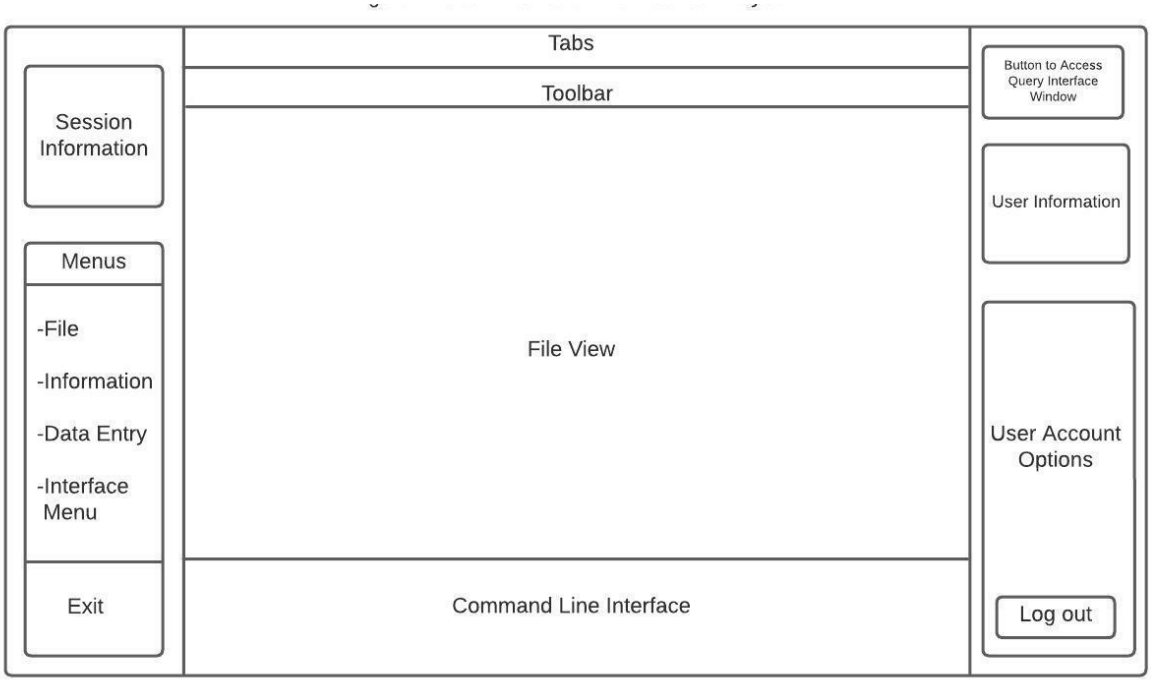


As shown in Figure 9, the log-in panel will have the following fields: username and password. In addition, the panel will accommodate a log-in button that will check if there is a match with the entered credentials and redirect to the main screen layout shown in Figure 11. If the credentials do not match, the user will be displayed an error message. Password reset button will allow users to reset their passwords via their email addresses if they do not remember their credentials. (See 5.2.2)

6.2 Main User Interface

Figure 11 represents a mockup for the main screen layout that will be used mainly by the users in the SCA tool.

Figure 11: User Interface - Main Screen Layout (adapted from the SRD document)



As shown in Figure 11, the main screen will have the following sections: session information, menus, user information, file view, user account options, tabs, command line interface and exit and logout buttons.

6.2.1. Session Information

Session information will keep a record of the user's session and display the time elapsed in HH:MM: SS format along with the date in MM:DD: YYYY format.

6.2.2. Menu

Menu section will allow users to access files in the repository through the File item, access data, report and query options through Information item, create, update or delete an object through Data Entry item and access external data through Interface item.

6.2.3. Tabs

Tabs will allow easy switching between different files while the toolbar will provide an easy-to-use interface for editing tools for files. It will readjust the allocated width for each tab based on the number of tabs open.

6.2.4. File View

File view section will take up the center of the screen, and display the file being worked on by the user. When there is no open file, FileView will be blank and display a solid white screen.

6.2.5. Command Line Interface

Command line interface will allow users to interact with the repositories through the terminal using the commands. This process will require validation of every command. Error messages will be displayed if the command is not identified.

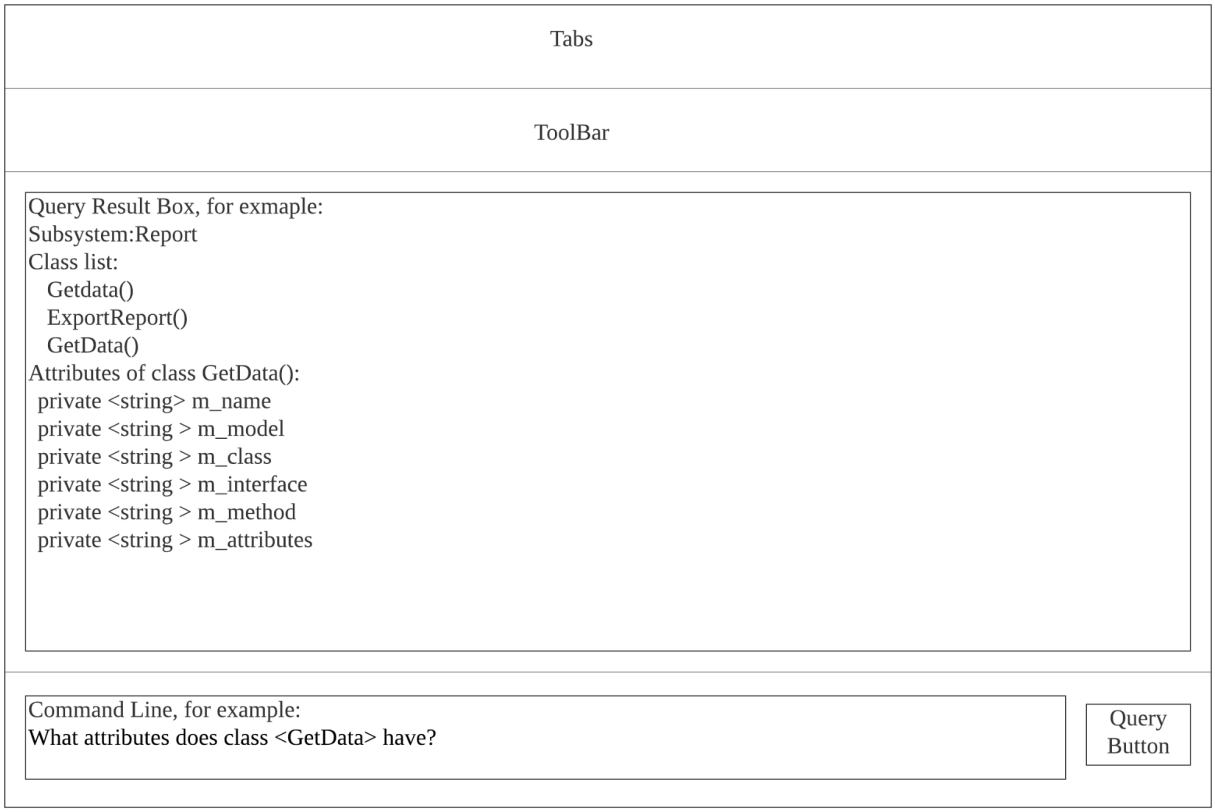
6.2.6. User Information and User Account Options

User information section will accommodate logged-in user's information such as name, last name, id and role whereas user account options will allow users to manage their accounts such as changing passwords. Log out button will log out of the user's account and return to the login panel while the exit button will shut down the tool entirely. Both processes will require an extra step of confirmation.

6.2.7. Query Interface Button

Query interface button will redirect the user to a new window dedicated to a complete query interface where they will be able to make different types of queries.

Figure 12: User Interface - Query Screen Layout



6.2.7.1 Tabs

Tabs will allow easy switching between different files. It will readjust the allocated width for each tab based on the number of tabs open.

6.2.7.2 Toolbar

the toolbar will provide an easy-to-use interface for editing tools for files. Users will be able to access other interfaces such as report and data entry without backing to the main interface.

6.2.7.3 Query Result Box

Initially, the query result box will display a list of subsystems automatically. Then the user can enter the command to query classes in one of the subsystems. And then the user can query the attributes or methods in one of the classes known by the former step and so on. If there isn't any error during the query process, the results will be displayed in the query result box in a certain sequence. Otherwise, error messages will be displayed instead.

6.2.7.4 Command Line

Command line will allow users to type commands directly to interact with the repositories. The command should be precise because the query subsystem generally yields small amounts of information.

If the user is not familiar with the system, he needs to query step by step. For example, if the user wants to query the attributes in a class, he should type" What classes does subsyem <Report> have?" to know the class list for the subsystem. Then the user type" What attributes does class <GetData> have" to get attributes. If the user is familiar with the system, he can have access to the attributes directly by typing "What attributes does class<GetData> have in subsystem <Report>".

6.2.7.5 Query Button

Once a user clicks the query button, the process of validation is required for every command. Error messages will be displayed if the command is not identified.